Systems Engineer Challenge (V2.0)

For clarity, the normally invisible newline character (ASCII 0x0A) is represented with <NL>.

Implement a simple POSIX/Unix shell (similar to bash).

The shell should support the following features at minimum:

- The shell should print "`$ `" on standard error (stderr) as a prompt. Commands should be read from standard input (stdin), and are delimited by newline (ASCII 0x0A). Command tokens are delimited by one or more spaces, with the first token being the command to invoke and the other tokens being the arguments to pass to the command. Extra spaces before the first token and after the last token should also be stripped.

- The shell should be able to run commands by their absolute paths:
```
$ /bin/ls -l /usr/share<NL>
total 12
drwxr-xr-x 1 root root    20 Mar 27  2018 accountsservice
drwxr-xr-x 1 root root 26672 Jan  3 13:19 aclocal
drwxr-xr-x 1 root root   608 Dec 13 17:34 aclocal-1.16
drwxr-xr-x 1 root root    30 Apr 20  2020 acpi_call
...
```

- The shell should be able to run commands present in the environment variable `$PATH`:
```
$ cat /proc/mounts<NL>
proc /proc proc rw,nosuid,nodev,noexec,relatime 0 0
sys /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
...
```

- The shell should handle single quotes and double quotes. No backslash escape handling is required. If quotes are mismatched, print an error. Errors should be printed on standard error (stderr).
```
$ /usr/bin/printf "The cat's name is %s.\n" 'Theodore Roosevelt'<NL>
The cat's name is Theodore Roosevelt.
$ /usr/bin/printf "Missing quote<NL>
error: mismatched quotes
```

- The shell should handle the `cd` built-in command to change the current directory. It should take a single argument: the absolute or relative path of the directory to change into. If the command fails, it should print an error. Errors should be printed on standard error (stderr).
```
$ cd /sys<NL>
$ ls<NL>
block  bus  class  dev  devices  firmware  fs  hypervisor  kernel  module
power
$ cd /nonexistent<NL>
error: cd failed
```

- The shell should handle the `exit` built-in command to exit and return control to its caller.
- The shell should exit and return control to its caller when it receives end of file (EOF) on its standard input.
- If a command exits with a non-zero exit code, the shell should print the exit code. The exit code should be printed on standard error (stderr).
  ```
  $ head /nonexistent<NL>
  head: cannot open '/nonexistent' for reading: No such file or directory
  error: command exited with code 1
  ```
- The shell should be written in C, C++, or Rust.
- Do not use any external libraries.
- Notwithstanding out-of-memory errors or external signals, the shell should never crash, even if it receives invalid input. A crash is defined as the program exiting for any reason other than `main()` returning or `exit()` being called.
- The shell may optionally error if the command line is longer than 1000 characters (including the terminating newline). However, this should not crash the shell.
- The shell may optionally error if the command line contains more than 100 arguments. However, this should not crash the shell.
- The error messages may be formatted in any format of your choosing.
- Any POSIX shell details not specified in this specification should be omitted (e.g. signal handling, pipes, redirection, environment variable expansion, line editing, history, etc.)

Test cases

Normal text is output from the shell itself on standard error (stderr).
Bold text is user input.
Red text is command output.

```
$ cd /sys<NL>
$ ls<NL>
block  bus  class  dev    devices  firmware  fs  hypervisor  kernel  module  power
$ <NL>
$ <NL>
$ /bin/ls -l<NL>
total 0
drwxr-xr-x   2 root root 0 Apr 22 18:47 block
drwxr-xr-x  46 root root 0 Apr 22 18:47 bus
drwxr-xr-x  73 root root 0 Apr 22 18:47 class
drwxr-xr-x   4 root root 0 Apr 22 18:47 dev
drwxr-xr-x  27 root root 0 Apr 11 20:55 devices
drwxr-xr-x   6 root root 0 Apr 11 20:55 firmware
drwxr-xr-x  11 root root 0 Apr 11 20:55 fs
drwxr-xr-x   2 root root 0 Apr 22 18:47 hypervisor
drwxr-xr-x  15 root root 0 Apr 11 20:55 kernel
drwxr-xr-x 306 root root 0 Apr 22 18:47 module
drwxr-xr-x   3 root root 0 Apr 22 18:47 power
$ /bin/echo  extra    spaces    will    be    removed<NL>
extra spaces will be removed
$ echo "but    not    if    they're    in    quotes"<NL>
but    not    if     they're      in     quotes
$   cd    '/proc/sys'<NL>
$ pwd<NL>
/proc/sys
$ mkdir '/tmp/"hello"'<NL>
$ cd '/tmp/"hello"'<NL>
$ pwd<NL>
/tmp/"hello"
$ cd ..<NL>
$ pwd<NL>
/tmp
$ false<NL>
error: command exited with code 1
$ /bin/sh -c 'exit 7'<NL>
error: command exited with code 7
$ exit<NL>
```

Extra notes

- In C/C++, child processes may be spawned using `fork()` and `execve()`. `posix_spawn()` is also an option. `wait()` or `waitpid()` may then be used to wait for the child process to exit.
- `execvp()` and `posix_spawnp()` additionally take care of the $PATH lookup for you.
- In Rust, `std::process::Command` may be used to spawn child processes.